Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Declan McPartlin

Inteligentni agent z omejenimi viri v dinamični računalniški igri

magistrsko delo

študijski program druge stopnje
računalništvo in informatika

Mentor: izr. prof. dr. Marko Robnik Šikonja

Ljubljana, 2014

Izjava o avtorstvu magistrskega dela

Spodaj podpisani Declan McPartlin sem avtor magistrskega dela z naslovom:

Inteligentni agent z omejenimi viri v dinamični računalniški igri

S svojim podpisom zagotavljam, da:

- sem magistrsko delo izdelal samostojno pod mentorstvom izr. prof. dr. Marka Robnika Šikonje,
- so elektronska oblika magistrskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko magistrskega dela,
- soglašam z javno objavo elektronske oblike magistrskega dela v zbirki "Dela FRI".

V Ljubljani, 11. marca 2014 Podpis avtorja:

Zahvaljujem se mentorju, izr. prof. dr. Marku Robniku Šikonji, as. Tomu Vodopivcu, as. Bojanu Klemencu in Piji Režek, kot tudi prijateljem in družini.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
RL	Reinforcement Learning	spodbujevano učenje
DP	Dynamic Programming	dinamično programiranje
MCM	Monte Carlo Methods	metode monte carlo
TDL	Temporal Difference Learning	učenje s časovnimi razlikami
SNARC	Stochastic Neural Analog Reinforcement Calculator	stohastični nevronski analogni spodbujevani kalkulator
GPS	General Problem Solver	splošni reševalec problemov
UI	Artificial Intelligence	umetna inteligenca
HMM	Hidden Markov Model	skriti markovski model
SOAR	State, Operator And Result	stanje, operator in rezultat (struktura)
MENACE	Matchbox Educable Naughts and Crosses Engine	poganjalnik za učljivo škatlico s križci in krožci
GLEE	Game Learning Expectimaxing Engine	poganjalnik za učenje iz iger po načelu expectimax
LMS	Least Mean Squared	najmanjše kvadratno povprečje
PUN	Trial and Error Learning	učenje iz napak
GPI	Generalized Policy Iteration	posplošena iteracija strategije
FSM	Finite State Machine	končni avtomati

Povzetek

Z inteligentnimi agenti se dandanes srečujemo dnevno – ko uporabljamo medmrežje, ko letimo z letalom ali kadar igramo računalniške igre. Vsi takšni agenti imajo mnogo skupnih lastnosti, nekateri za svoje delovanje potrebujejo veliko računalniških virov. V nekaterih primerih so viri za učinkovito delovanje agentov omejeni, kot npr. na mobilnih napravah. V takšnih primerih potrebujemo čim manj zahtevnega, a hkrati dovolj pametnega agenta, ki še zmeraj verno posnema človeško obnašanje.

Čeprav se popularnost večigralnega načina v igrah v realnem času veča na stacionarnih konzolah, na mobilnih napravah hitra in zagotovljena medmrežna povezava ni zmeraj mogoča. Zaradi tega se večigralni način igre v realnem času na mobilnih napravah ne uporablja tako množično – bolj pogosto se uporabljata asinhroni večigralni način igre in enoigralni način. V enoigralnem načinu igre mora simulirani nasprotnik za dobro uporabniško izkušnjo igralca čim bolje posnemati človeka. Potrebujemo torej inteligentnega agenta, ki ima sposobnost učenja z omejenimi viri. V magistrski nalogi na primeru dinamične igre prikažemo nekaj različnih algoritmov umetne inteligence, ki poskušajo doseči ta cilj.

Kot primer implementacije inteligentnega agenta smo uporabili igro Knoxball. Knoxball je dinamična igra, ki je mešanica med nogometom in zračnim hokejem. Agenta smo zgradili na podlagi spodbujevanega učenja, ki z razliko od nadzorovanega učenja, ne potrebuje označenih podatkov, saj se uči iz lastnih izkušenj. Agent ne pozna pravil igre, uči se iz izkušenj in na podlagi povratnih informacij iz okolja.

Učinkovitost našega inteligentnega agenta je naraščala z nabranimi izkušnjami – sčasoma se je izkazal kot uporaben. Na koncu testiranja je brez večjih težav premagal agenta, izdelanega na podlagi končnega avtomata, ki je služil za osnovo. Dobre rezultate je dosegel tudi proti človeškim igralcem.

Abstract

Nowadays we encounter intelligent agents on a daily basis, when we use the internet, take a flight, or when we play video games. All these agents have many things in common, some of them require a lot of resources. In some situations, for instance in mobile devices, we might not have the desired resources to power such an agent. In these cases we need less demanding yet intelligent agents, that still closely mimic human behaviour.

Although real time multiplayer gaming is growing in popularity on stationary consoles, a secure and fast internet connection is not always available on handheld devices, which makes real time multiplayer gaming less attractive on handheld devices. Because of this, many mobile games tend to opt for asynchronous multiplayer modes or put more focus on a single player mode. To create the best user experience in single player mode we want to make the opponent seem human. We need an intelligent agent that can learn with limited resources. Therefore, we need to combine several different artificial intelligence approaches.

We chose Knoxball, a mobile game, as an example environment to implement our agent. Knoxball is a dynamic game that is a mix between soccer and air hockey. Our agent is based on a combination of reinforcement learning algorithms. Reinforcement learning doesn't need labelled data to start learning, it gathers the data on its own. The agent was never taught the rules of Knoxball, it learns from experience, using feedback from the environment.

After the initial learning period, our agent has proved to be versatile and able to play Knoxball fairly well. The agent's efficiency grew with time. At

the end of testing, our agent was able to beat the finite state machine based agent. The agent faired well against human players too.

Poglavje 1

Uvod

Inteligentni agenti se že dolgo uporabljajo v računalniških igrah in tudi v drugih sistemih. V igrah se jih največkrat uporablja za računalniško vodene like, ki posnemajo človeško ali živalsko obnašanje: lovijo plen, iščejo izhod, se na čim bolj učinkovit način premikajo na drugo lokacijo, planirajo napad, kažejo čustva ali preizkušajo igralčevo znanje. Algoritme, s katerimi to obnašanje implementiramo, imenujemo umetna inteligenca [4].

Umetna inteligenca je sestavni del iger že od časa Pac-Mana, ko so računalniške igre vključevale zelo primitivne agente, implementirane s končnim avtomatom. Te še danes srečamo v igrah na mobilnih napravah, vedno bolj prisotne pa so nevronske mreže. Agente, ki so se zmožni učiti, se le izjemoma uporablja v arkadnih igrah, saj načeloma obstaja le končno število kombinacij različnih potez in si agent lahko hitro sestavi bazo podatkov, kateri napadi so učinkoviti. Pri drugih zvrsteh iger se večinoma inteligentnih agentov ne splača implementirati, saj so računsko preveč požrešni [7].

V magistrski nalogi izhajamo iz igre Knoxball, ki je simulacija nogometa z nekaterimi elementi zračnega hokeja. Knoxball se igra v realnem času. Igralec se premika po igrišču s ciljem, da spravi žogo v nasprotnikovo mrežo. Stopnja težavnosti igre je odvisna od zmožnosti nasprotnika. Človeški igralec se izboljšuje s časom, medtem ko agent, ki se ne uči, hitro postaja neenakovreden nasprotnik človeku. Čeprav igra izgleda enostavno, je število strategij do

zadetka zelo veliko. Boljši kot je nasprotnik, bolj mora biti igralec učinkovit, spreten in izurjen, da ga lahko premaga.

Podobne igre uporabljajo le osnovne pristope umetne inteligence. Nekatere uporabljajo inteligentne agente – nasprotnike, ki se učijo, vendar uporabljajo informacije shranjene na strežniku, kjer izvajajo tudi večino računanja. Večina računalniških nasprotnikov, ki se izvajajo lokalno na mobilnih napravah, uporablja agente, ki so končni avtomati. Nekateri so sicer kompleksni, vendar se niso zmožni naučiti bolj optimalnih taktik.

Agenta s končnim avtomatom smo implementirali tudi za našo igro Knoxball. Problem pri tem pristopu je, da agent v enakih okoliščinah vedno enako odreagira, kar je nenaravno in slabo vpliva na igralnost igre. Uvedli smo nekaj naključnosti pri premikanju igralca, vendar je bila razlika v odzivu majhna. Z vidika igralca se agent ponavlja. Če dodamo več naključnosti, je igralec tako nepredvidljiv, da je oslabljen in nenaraven. V okviru magistrske naloge smo želeli implementirati agenta, ki se je zmožen učiti iz prejšnjih dogodkov v realnem času, torej z omejenimi viri na mobilni platformi.

Potrebujemo algoritme strojnega učenja, ki omogočajo agentu, da sčasoma izboljšuje obnašanje brez predhodnega znanja. Agent mora znati izkoristiti znanje, ki ga je pridobil z nabiranjem novih, morda bolj uporabnih izkušenj. Agent ne sme ponavljati enakih napak, poleg tega mora delovati z omejenimi viri. Pristop z nadzorovanim učenjem potrebuje predhodno označene podatke in ima jasno ločnico med učenjem in izkoriščanjem znanja, zato za naš problem ni primeren. Za rešitev našega problema smo uporabili spodbujevano učenje.

Spodbujevano učenje je učenje iz preteklih izkušenj s ciljem maksimizirati uspešnost pri doseganju vnaprej določenih ciljev. Agent se odloči, kakšna bo njegova naslednja akcija na podlagi izkušenj, ki jih je nabral do tistega trenutka. Akcije, ki ustvarjajo stanje okolja, v katerem je agent zmožen dosegati svoje cilje, bodo večkrat izbrane tudi v prihodnje, saj prinašajo večje nagrade. Spodbujevano učenje smo izbrali zaradi fleksibilnosti in zmožnosti za učenje. Človeku, ki igra igro proti nasprotniku, ki se neprestano izboljšuje,

se bo igra zdela bolj zanimiva.

Podobne igre, kot so Padkick in Soctics za iOS ter Haxball v namiznem brskalniku, nimajo implementiranih inteligentnih agentov, saj nimajo enoigralskega načina igre. Na igrišču so samo človeško vodeni igralci.

Nalogo sestavlja šest poglavji. V drugem poglavju prikažemo pregled sorodnih del in zgodovino umetne inteligence, spodbujevanjega učenja in z iger z umetno inteligenco. V tretjem poglavju podrobneje opišemo delovanje spodbujevanega učenja. V četrtem poglavju sledi podroben opis implementacije agenta s spodbujevanim učenjem. V petem poglavju opišemo način evalvacije rezultatov naših agentov. V zadnjem poglavju strnemo zaključne ugotovitve in predstavimo možnosti nadaljnjega razvoja.

Poglavje 2

Pregled področja

V tem poglavju pregledamo, kako se je razvijala umetna inteligenca skozi prejšnja desetletja, in največje dosežke te veje v zadnjem času. Del poglavja smo povzeli po Russllovi in Norvigovi knjigi Artificial Inteligence – A modern Approach [16]. Podrobneje pregledamo še zgodovino veje strojnega učenja, imenovano spodbujevano učenje, kar smo povzeli po Suttonovi in Bartovi knjigi Reinforcement Learning: An introduction [22]. Na koncu preletimo razvoj umetne inteligence v industriji računalniških iger.

2.1 Umetna inteligenca

Veda o inteligentnih agentih in umetna inteligenca nista nekaj novega, čeprav je še zmeraj stvar debate, kaj točno pomeni umetna inteligenca.

Delo Warrena McCullocha in Walterja Pittsa iz leta 1943 je splošno sprejeto kot prvo delo s področja umetne inteligence. Njuno delo izhaja iz treh virov: znanja o osnovni fiziologiji in delovanju nevronov v možganih, formalne analize propozicijske logike ter Turingove teorije o računanju. Predlagala sta model umetnih nevronov, ki so bili prižgani ali izklopljeni glede na ustrezno stimulacijo sosedov. Namigovala sta na to, da bi se lahko takšen model tudi učil. Donald Hebb je pri tem demonstriral enostaven zakon ažuriranja jakosti povezav med nevroni. Njegov zakon, zdaj poimenovan Hebbovo učenje,

ostaja vpliven model še danes.

Ena najbolj vplivnih oseb v začetku razvoja umetne inteligence je bil Alan Turing. O svojih vizijah je predaval v Londonskem matematičnem združenju. O umetni inteligenci je pisal tudi v članku Computing Machinery and Intelligence [18]. Predstavil je preizkus, danes imenovan Turingov test inteligence, strojno učenje, genetske algoritme in spodbujevano učenje. Predstavil je tudi idejo programa – otroka, kjer je dejal: »Namesto da se trudimo izdelati program, ki simulira odraslega, zakaj ne bi raje izdelali programa, ki bi simuliral otroka?« [18].

Leta 1951 je John McCarthy v Dartmouth Collegu zbral deset največjih poznavalcev umetne inteligence. Dva meseca so skupaj razvijali različne ideje. Najbolj uspešna sta bila Newell in Simon, ki sta ustvarila sistem Logic Theorist. Simon je trdil: »Izdelali smo računalniški program, ki je zmožen razmišljati nenumerično in s tem rešili problem zavesti.« Čeprav je program dejansko našel dokaz za eno izmed teorij iz Principia Mathematica, krajšega od zapisanega, je bil Newellov in Simonov članek o njunem programu zavrnjen v Journal of Symbolic Logic [16].

Rochester in sodelavci so ustvarili nekaj prvih pravih programov umetne inteligence. Gelernter je ustvaril program Geometry Theorem Prover, ki je lahko dokazoval težke teoreme [6]. Na poti do tega rezultata je dokazal, da ni res, da računalniki lahko delajo samo to, kar jim je naročeno. Njegov program se je hitro naučil, kako igrati igro boljše kot njegov lastnik.

Leta 1958 je McCarthy definiral jezik lisp, ki je postal prevladujoči programski jezik za UI (umetno inteligenco) za naslednjih 30 let. Isto leto je izdal članek, imenovan »Programs with Common Sense« [10], v katerem je opisal hipotetičen program »Advice Taker«, ki je znan kot prvi popolni sistem umetne inteligence. Tako kot programa »Logic Theorist« in »Geometry Theorem Prover« je bil tudi »Advice Taker« zamišljen tako, da uporablja znanje za rešitve nekaterih problemov, ampak z razliko od ostalih naj bi ta program uporabil celotno splošno znanje, zato McCarthy pravi, da je program popolni sistem umetne inteligence [16].

Minsky je vodil skupine študentov, ki so se osredotočili na specifične problemske domene, ki navidezno potrebujejo inteligenco za reševanje. Te omejene svetove so poimenovali mikrosvetovi. Kot primer je program »Analogy« Toma Evansa iz leta 1968 reševal probleme iz domene geometrijske analogije, tipične za fakultetne predmete. Najbolj znan mikrosvet je bil svet kock, v katerem je robotska roka postavljala kocke [15]. Robotska roka je lahko premikala eno kocko naenkrat.

Uspešen ekspertni sistem je bil MYCIN, ki so ga uporabljali za diagnozo krvnih infekcij. Sistem se je izkazal kot natančen, saj se je lahko primerjal s točnostjo ekspertov ter napovedoval precej boljše kot mladi zdravniki. Sistem je deloval na podlagi približno 450 pravil [20].

Skriti markovski model (angl. Hidden Markov model, HMM) je dominiral na področju umetne inteligence v 80. letih prejšnjega stoletja. Izkazal se je kot učinkovit pri pretvarjanju zvoka v besedilo, čeprav ni pričakovati, da naši možgani delujejo na podoben način. HMM se generirajo na podlagi velikih zbirk z realnimi govornimi podatki.

V 90. letih prejšnjega stoletja je ponovno postala popularna ideja o splošnem inteligentnem agentu. SOAR, delo Newella, Lairda in Rosenblooma, je eden najbolj znanih primerov arhitekture popolnega agenta [16].

Popularnost inteligentnih agentov je močno narastla zaradi pojavitve medmrežja. Tehnike umetne inteligence se uporabljajo v veliko različnih orodjih na medmrežju, kot so iskalniki, svetovalni sistemi itd.

Leta 1997 je IBM-ov Deep Blue, ki so ga začeli razvijati leta 1985, premagal takrat večkratnega svetovnega prvaka v šahu Kaspara. V zadnjih letih računalnik v šahu prepričljivo zmaguje [16].

2.2 Spodbujevano učenje

Vejo strojnega učenja, imenovano spodbujevano učenje (angl. reinforcement learning), bomo zgodovino podrobneje predstavili, ker je najbolj obetavna za rešitev našega problema. Za razliko od nadzorovanega učenja spodbujevano

učenje ne potrebuje označenih podatkov že na začetku. Poleg tega agent na podlagi spodbujevanega učenja neprestano izboljšuje rezultate in obnašanje zato, ker s časom nabira izkušnje in izkorišča nabrano znanje.

V času začetkov spodbujevanega učenja je obstajalo več vej, iz katerih se je kasneje razvilo današnje spodbujevano učenje. Ena izmed vej se je razvila iz psihologije o učenju pri živalih. Druga veja izhaja iz teorije optimalnega nadzora, čeprav ta veja na začetku ni bila osredotočena na učenje. Tretja veja se ukvarja z metodami časovnih razlik (angl. temporal difference).

2.2.1 Optimalen nadzor

Terminologija »optimalen nadzor« izhaja iz 50. let prejšnjega stoletja kot opis formuliranja krmilnika, ki minimizira dinamičnost obnašanja sistemov skozi čas. Eno izmed prvih rešitev takega problema je predstavil Richard Bellman s sodelavci. Rešitev uporablja koncept stanj v dinamičnem sistemu in »funkcijo z optimalno vrednostjo«, funkcijsko enačbo, zdaj imenovano Bellmanova enačba. Razred metod, s katerimi se take enačbe rešujejo, je znan kot dinamično programiranje. Bellman je predstavil tudi diskretno stohastično verzijo problema optimalnega nadzora, znano kot markovski odločitveni problem. Omenjeni koncepti so podlaga za spodbujevano učenje.

2.2.2 Učenje iz napak

Glavna veja spodbujevanega učenja izhaja iz psihologije, njena glavna ideja je preizkušanje in učenje iz napak in uspehov. V psihologiji je več takšnih teorij o učenju iz napak (angl. Trial and Error). Eden prvih, ki je idejo takšnega načina učenja predpostavil, je bil Edward Thorndike. Poglavitna ideja je, da so dejanja, ki so se končala z uspešnim izidom, večkrat izbrana kot tista, ki so se končala s slabim izidom. Thorndike je opisano idejo imenoval zakon učinka (angl. Law of Effect).

V 60. letih prejšnjega stoletja se je prvič uporabil pojem spodbujevano učenje (npr. Waltz and Fu, 1965; Mendel, 1966; Fu, 1970; Mendel and

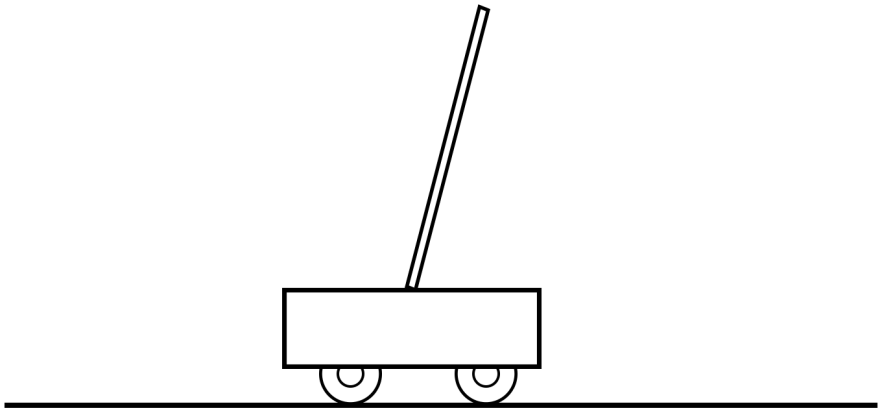
McClaren, 1970). Zanimiv članek je napisal Minsky leta 1961, imenovan »Steps toward Artificial Intelligence«. V njem je opisoval problem dodeljevanja nagrade, ko je do rezultata še veliko korakov [14].

Leta 1963 je John Andreae ustvaril sistem STeLLA, ki se je učil s pomočjo učenja iz napak v svojem okolju. Sistem je imel interni model sveta in kasneje »notranji monolog« za reševanje problema skritih stanj.

Donald Michie je leta 1961 in 1963 opisal osnoven sistem učenja iz napak za učenje pri igri tic-tac-toe, imenovan MENACE (angl. Matchbox Educable Naughts and Crosses Engine). Sestavljen je bil iz škatlic za vsako možno pozicijo v igri. Vsaka škatlica je imela neko število barvnih kroglic. Vsaka možna poteza je bila predstavljena z eno barvno kroglico z drugačno barvo. Iz škatlice je sistem naključno izbral eno kroglico in s tem določil odločitev za naslednjo potezo sistema. Na koncu igre je sistem kroglico odstranil ali dodal glede na to, ali je dobil nagrado ali kazen – glede na končni rezultat [5].

Leta 1968 sta Michie in Chambers opisala še en sistem za igranje in spodbujevano učenje pri igri tic-tac-toe, imenovan GLEE (angl. Game Learning Expectimaxing Engine). Sistem je uporabil upravljavca s spodbujevanim učenjem, imenovan BOXES. BOXES je imel nalogo, da se nauči držati v ravnovesju palico na premikajočem se vozičku, pri tem pa se oglasi signal neuspeha, če palica pade ali pa voziček pride do konca poti, kot je prikazano na sliki 2.1. Njuno delo se je izkazalo kot zelo vplivno na kasnejša dela (Barto, Sutton, and Anderson, 1983; Sutton, 1984). Michie je velikokrat poudaril, da imata preizkušanje in učenje iz napak ključno vlogo pri umetni inteligenci [12].

Widrow, Gupta in Maitra so modificirali Widrowov in Hoffov algoritem LMS (angl. Least Mean Squared) v algoritem, ki je postal zakon spodbujevanega učenja in se je znal učiti iz signalov uspeha ali izgube namesto iz učnih primerov. Algoritem so analizirali in pokazali, da se je bil zmožen naučiti igre blackjack. Takšen način učenja so poimenovali »selektivna strnenska prilagoditev« (angl. Selective bootstrap adaptation) in ga opisali kot učenje



Slika 2.1: Voziček, ki se uči, kako balansirati palico s premikanjem.

s kritikom namesto z učiteljem.

Raziskave o učečem se avtomatu so imele bolj neposreden vpliv na vejo učenja iz napak, kar je vodilo v moderne raziskave spodbujevanega učenja. Učeči se avtomati so zelo osnovni nizko-spominski stroji za reševanje problemov, kot so »n-ročni bandit«. Učeči se avtomat izvira iz Rusije (delo Tselina 1973) in se je od takrat še razvijal (Narendra 1974, Thathachar 1989).

Leta 1975 je John Holland predstavil splošno teorijo o adaptivnih sistemih na podlagi izbranih načel. Njegovo zgodnje delo je obsegalo preizkušanje in učenje iz napak, evolucijske metode in »n-ročni bandit«. Leta 1986 je predstavil klasificijske sisteme, prave sisteme s spodbujevanim učenjem, ki vključujejo funkcijo vrednosti. Ključni del Hollandovega klasifikacijskega sistema je genetski algoritem, evolucijska metoda, katere vloga je, da razvije uporabne agente. Klasifikacijske sisteme je razvijalo veliko raziskovalcev, toda genetski algoritmi so dobili več pozornosti [22].

Posameznik, ki je najbolj pripomogel k oživljanju učenja iz napak v spodbujevanem učenju, je bil Harry Klopf [22]. Ugotovil je, da so bili bistveni vidiki adaptivnega obnašanja izgubljeni, ko so se raziskovalci skoraj izključno osredotočili na nadzorovano učenje. Kar je po njegovem mnenju manjkalo, so bili hedonistični vidiki obnašanja, ki se trudijo doseči rezultat tako, da

usmerjajo okolico proti želenemu razpletu in stran od nezaželenega razpleta.

2.2.3 Učenje s časovnimi razlikami

Tretja veja spodbujevanega učenja se imenuje učenje s časovnimi razlikami (angl. Temporal Difference learning, TDL).

To učenje delno izvira iz psihologije učenja pri živalih, še posebej v smislu sekundarnih ojačevalcev (angl. secondary reinforcers). Sekundarni ojačevalec je spodbudnik, ki je povezan s primarnim ojačevalcem, kot so recimo hrana ali bolečina, zaradi tega prevzame podobne ojačevalne lastnosti. Minsky (1954) je opazil, da je to psihološko načelo lahko pomembno za sisteme umetne inteligence. Arthur Samuel (1959) je predstavil in implementiral metodo učenja, ki je vključevala ideje TDL kot del programa, ki je igral damo (angl. checkers) [17].

Leta 1972 je Klopf združil učenje iz napak s komponento na podlagi TDL. Klopfa je zanimalo učenje v velikih sistemih in se je zato zanimal za ideje o lokalni ojačitvi, ker bi komponente nekega večjega učnega sistema lahko ojačale ena drugo. Razvil je idejo posplošene ojačitve (angl. Generalized Reinforcement), kjer bi vsaka komponenta opazovala svoje vhode v smislu spodbujevanega učenja: spodbujevalen vhod kot nagrada in zaviralni vhodi kot kazen. V retrospektivi je delo sicer dlje od TDL kot Samuelovo delo, je pa Klopf povezal idejo učenja iz napak s psihologijo učenja pri živalih.

Sutton je razvijal ideje Klopfa, zlasti povezave s teorijo učenja pri živalih, in opisoval zakone učenja pri spremembah v časovno zaporednih napovedih. Skupaj z Bartom sta izpilila omenjene ideje in razvila psihološki model klasičnega pogojevanja (angl. conditioning) na osnovi TDL. Schultzov, Dayanov in Montagueov povzetek povezuje TDL z idejami iz nevroznanosti [19].

Veji TDL in učenje z optimalnim nadzorom sta se povezala leta 1989 z Watkinsovim razvitjem Q-učenja. Omenjeno delo razširja in vključuje prejšnje delo v vseh treh vejah spodbujevanega učenja. Werbo (1987) je prispeval k temu z združitvijo učenja iz napak in dinamičnega programiranja. V zadnjih letih je vedno več prispevkov s področja spodbujevanega učenja [22].

V naslednjem podpoglavju predstavimo zgodovino razvoja UI v računalniških igrah.

2.3 Zgodovina UI v računalniških igrah

Poleg iger Pong in Space Invader je bila igra Pac-Man, ki je izšla leta 1979, ena izmed prvih iger, ki je vsebovala inteligentne agente, ki so dajali igralcu občutek, da ima razmišljujočega nasprotnika, ki kuje zaroto proti igralcu. Nasprotniki so se premikali na podoben način kot igralec in mu otežili delo. Agenti iz igre Pac-Man so uporabljali preprost avtomat (angl. State machine). Agenti, ki so duhove vodili, so bili ali v stanju sledenja igralcu ali v stanju bežanja od igralca. Na vsakem križišču je agent z neko verjetnostjo izbral najbolj optimalno pot, sicer pa je pot izbral naključno.

Leta 1987 je SEGA izdala igro Golden Axe. V igri so implementirali agente, ki so počakali na igralca, nato pa ga takoj napadali ali pa so se premaknili na drugo stran in se šele takrat spravili nanj. Agenti so bili še zmeraj precej enostavni.

Podjetje Rare Ltd. je izdalo igro Goldeneye 007 leta 1997. Igra je pokazala precejšen napredek UI v računalniških igrah. Čeprav je bilo delovanje agentov znotraj igre osnovano z uporabo končnih avtomatov, so dodali sistem simulacije čutov (angl. Sense Simulation System). Agenti so dajali občutek, da so znali opazovati ostale igralce in primerno reagirati, če je bil kakšen izmed njihovih tovarišev ustreljen. Sistem simulacije čutov je bil večkrat uporabljen v drugih igrah tega časa, kot so Thief: The Dark Project (Looking Glass Studios Inc., 1998), Metal Gear Solid (Konami Corporation, 1998) itd.

Igra Warcraft (izdal jo je Blizzard leta 1994) je bila ena izmed prvih iger, v katero so vključili agente, ki so uporabljali napredne algoritme za iskanje poti (angl. Path Finding).

UI v obliki modela čustev vojakov je prikazala igra simulacije bojišč Warhammer: Dark Omen, ki jo je izdal Mindscape leta 1998. Igra je bila tudi

ena prvih, ki je uporabnikom prikazala premikanje enot v robustni formaciji.

Igra, ki je temeljila na UI, je bila Creatures (Cyberlife Technology Ltd., 1997). Ta igra ima enega najbolj kompleksnih sistemov UI v igrah. Vsak lik ima svoje »možgane« v obliki nevronske mreže. Živali v igri, poimenovane »Norni«, imajo lastnosti agentov, ki so se zmožni učiti. V tem smislu je igra Creatures še zmeraj edinstvena. Iger, ki imajo implementirane agente, ki se lahko učijo, je zelo malo [7], medtem ko pri igrah, kot so The Sims (Maxis Sofware, Inc., 2000) in Black and White (Lionhead Studios Ltd., 2001) igralnost igre močno temelji na UI.

V kombinaciji s končnim avtomatom se v igrah mnogokrat uporabljajo koncepti mehke logike. Mehka logika pripomore v situacijah, ko ni binarne izbire. Mehka logika skupaj s končnim avtomatom je bila implementirana v strelski igri Unreal, kjer agenti bežijo, ko izgubljajo, se skrijejo, če so poškodovani, vodijo igralca v zasedo itd. [7].

Dandanes je ogromno različnih primerov UI v igrah. Veliko iger še zmeraj uporablja zelo poenostavljen sistem UI, saj več od tega niti ne potrebujejo. Agenti v strelskih igrah uporabljajo več akademske UI kot druge veje UI [13]. Takšna UI se uporablja tudi za treniranje vojakov, npr. igro Full Spectrum Warrior (Pandemic Studios, 2004) so na začetku razvijali za potrebe vojske.

UI se vključuje tudi v športne igre, vendar imajo te igre posebne zahteve in veliko izzivov je še zmeraj nerešenih. Igre z vlogami (angl. Role Playing Games) imajo npr. težave z interakcijo z agenti pri pogovorih [13].

V naslednjem poglavju podrobneje opišemo, kako deluje spodbujevano učenje.

Poglavje 3

Opis učenja

Končni cilj naše naloge je implementirati agenta, ki bi s časom in z nabiranjem izkušenj igral bolje in na koncu morda celo prekašal človeškega igralca. Želimo agenta, ki se je zmožen učiti iz svojih napak.

Za našo nalogo potrebujemo koncept umetne inteligence, s katerim bi lahko implementirali agenta, ki se je zmožen prilagajati novim okoliščinam, se drugače obnašati v primeru neuspeha in iskati najbolj optimalno naslednjo potezo znotraj igre, ne da bi se ujel v lokalne ekstreme. Odločili smo se, da je najprimernejša rešitev za naš problem agent, implementiran s spodbujevanim učenjem.

Najprej opišemo osnove spodbujevanega učenja, nato pa podrobneje predstavimo glavne komponente spodbujevanega učenja in njihovo medsebojno delovanje. Predstavimo lastnost, ki je zelo pomembna v sistemih s spodbujevanim učenjem, imenovano markovska lastnost. Na koncu opišemo tri glavne načine implementacije spodbujevanega učenja.

3.1 Spodbujevano učenje

Spodbujevano učenje pomeni, da se učimo, kaj storiti oz. kako preslikati okoliščine v dejanje tako, da maksimiziramo numerično nagrado. Učencu ni predpisano, katero akcijo naj stori kot pri večini oblik strojnega učenja.

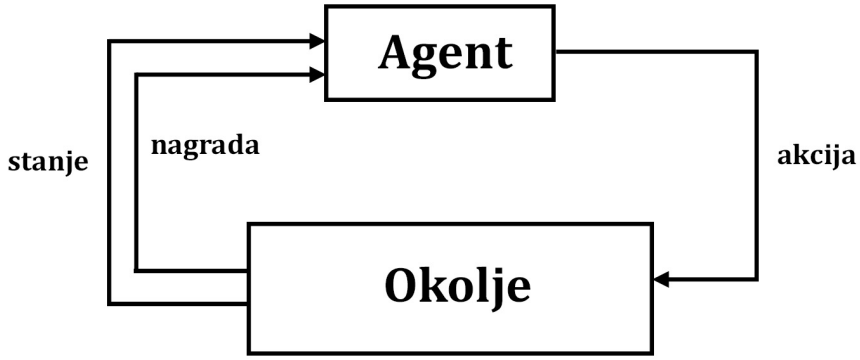
Namesto tega mora raziskovati, katere akcije vrnejo največjo nagrado, tako da jih izvaja. Najbolj zanimivi in zapleteni so primeri, ko akcije lahko vplivajo ne samo na takojšnjo nagrado, ampak tudi na naslednjo situacijo in preko tega na vse naslednje nagrade. Učenje iz napak in nagrada z zamudo sta dve najpomembnejši značilnosti spodbujevanega učenja.

Spodbujevano učenje se razlikuje od drugih načinov učenja v tem, da ni definirano z metodo učenja, ampak z opisom učnega problema. Katerakoli metoda, ki je primerna za reševanje tega problema, se smatra kot metoda spodbujevanega učenja. Agent se mora do neke mere zavedati stanja okolja in biti sposoben izvedeti za akcijo, ki lahko vpliva na okolje. Agent mora imeti tudi zastavljene cilje, ki so vezani na stanje okolja. Formulacija vključuje tri vidike:

- zaznavanje,
- ukrepanje,
- cilje.

Spodbujevano učenje ima drugačen način učenja kot nadzorovano učenje. Nadzorovano učenje temelji na detekciji vzorcev v statističnih podatkih z uporabo algoritma, kot je nevronska mreža ali linearna regresija. Nadzorovano učenje je učenje iz primerov, ki so podani in označeni vnaprej. Takšen način učenja je pomemben, ampak ni zadosten za učenje iz interakcij. V interaktivnih problemih je velikokrat nepraktično priskrbeti primere zaželenega obnašanja, ki so tako pravilni kot tudi zastopniški za vse situacije, v katerih bo agent moral ukrepati. V neznanem območju, kjer pričakujemo, da bo učenje najbolj uporabno, se mora agent učiti iz svojih izkušenj. Diagram na sliki 3.1 prikazuje agentovo sodelovanje z okoljem.

Specifičen problem, ki se pojavi pri spodbujevanem učenju in ne pri drugih vrstah strojnega učenja, je dilema **raziskovanje** (angl. exploration) ali **izkoriščanja** (angl. exploitation). Da agent pobere čim več nagrad, mora izvesti akcije,za katere je že spoznal, da so učinkovite pri ustvarjanju nagrade.



Slika 3.1: Medsebojna interakcija med agentom in okoljem.

Da odkrije še bolj učinkovite akcije, pa mora preizkusiti tudi akcije, ki jih prej še ni izbral. Agent mora izkoristiti, kar že ve, da pride do čim večje nagrade, toda hkrati mora raziskovati zato, da izbere čim bolj učinkovite akcije v prihodnosti. Agent s spodbujevanim učenjem mora kombinirati tako izkoriščanje kot raziskovanje, sicer ne doseže dobrega rezultata. Dilemo izkoriščanja in raziskovanja so matematiki intenzivno raziskovali več desetletij [22]. Te težave pri nadzorovanem učenju ni, saj je po navadi že definirano, kdaj raziskovati in kdaj izkoriščati znanje.

Pri mnogih drugih pristopih k strojnemu učenju se večji problemi razbijejo na manjše dele in se vsak podproblem rešuje posebej. Pri spodbujevanem učenju pa ciljno usmerjen agent smatra celoten problem za stohastično okolje. Vsi agenti s spodbujevanim učenjem imajo izrecne (angl. explicit) cilje. Lahko zaznavajo okolje in izbirajo akcije, ki vplivajo na okolje. Čeprav je na začetku okolje neznano in stohastično, se pričakuje, da se bodo akcije izvajale takoj. Če agent s spodbujevanim učenjem vključuje načrtovanje za naprej, mora določiti, na kakšen način bo kombiniral načrtovanje in izbiro akcije v realnem času ter tudi vprašanje, na kakšen način pridobi model okolja in kako ga izboljšuje.

V naslednjem podpoglavju pregledamo bistvene komponente, ki sestavljajo sistem s spodbujevanim učenjem.

3.2 Komponente sistema s spodbujevanim učenjem

Poleg agenta in okolja obstajajo še štirje glavni elementi sistema s spodbujevanim učenjem:

- strategija (angl. Policy);
- funkcija nagrade (angl. Reward function);
- funkcija vrednosti (angl. Value function);
- model okolja (angl. model of the environment).

Agentova **strategija** določi, na kakšen način se agent obnaša. Strategija predstavlja preslikavo med stanji okolja in akcijami, ki se lahko izvedejo v teh stanjih. Strategijo lahko predstavlja zelo preprosta funkcija ali pregledovalna tabela, lahko pa je predstavljena z zahtevno funkcijo. Strategija je v jedru agenta s spodbujevanim učenjem v smislu, da s strategijo določimo obnašanje agenta.

Funkcija nagrade definira cilj v zadanem problemu. Funkcija nagrade preslika vsako stanje okolja v realno število ali nagrado, ki določa zaželenost danega stanja. Edini cilj agenta s spodbujevanim učenjem je, da maksimizira skupno nagrado, ki jo prejme dolgoročno. Funkcija nagrade določa, katera stanja so za agenta slaba in katera so dobra. Ključno pri tem je, da agent ni zmožen spreminjati te funkcije, je pa osnova, ki določa spreminjanje strategija agenta. Funkcije nagrade so lahko stohastične.

Funkcija nagrade kaže, kaj je dobro takoj v naslednjem koraku, **funkcija vrednosti** določa, kaj je dobro na dolgi rok. Enostavno rečeno, funkcija vrednosti določa, kolikšno nagrado lahko agent pričakuje na dolgi rok iz določenega stanja. Nagradna vrednost opisuje takojšnjo nagrado agentu, medtem ko vrednost stanja upošteva možna stanja naprej in njihove nagrade. Nagradno vrednost bi lahko opisali kot občutek užitka in bolečine, medtem ko vrednost stanja predstavlja občutek zadovoljstva ali nezadovoljstva.

V tem smislu so nagrade primarni, vrednosti pa sekunarni odzivi. Brez nagrad ne bi bilo vrednosti. Edini razlog, zakaj računamo vrednosti, je maksimiziranje nagrad. Ko pride do odločitve, se naslanjamo na vrednosti in ne na nagrade. Izberemo akcije, ki vodijo do stanj z najvišjimi vrednostmi. Na žalost je težje vrednosti določiti kot nagrado. Nagrade dobimo iz okolja, vrednosti iz funkcije vrednosti pa moramo izračunati glede na izkušnje agenta. Ena najpomembnejših komponent sistema s spodbujevanim učenjem je metoda, ki učinkovito izračunava funkcijo vrednosti.

Funkcija vrednosti je lahko najpomembnejša komponenta sistema s spodbujevanim učenjem, vendar ni ključna. Lahko obstajajo sistemi s spodbujevanim učenjem, ki ne vključujejo funkcije vrednosti. Namesto tega uporabljajo genetske algoritme, genetsko programiranje in podobne funkcije. Omenjene metode delujejo znotraj strategije agenta, ne uporabljajo vrednosti stanj. Evolucijske metode imajo prednost pred sistemi s spodbujevanim učenjem s funkcijami vrednosti, če agent ne more točno zaznavati stanja svojega okolja.

Ena izmed razlik med uporabljanjem evolucijskih metod ali funkcij vrednosti je namreč ta, da evolucijske metode niso v neposredni interakciji z okoljem. Evolucijske metode ignorirajo pomembno lastnost: čeprav izbirajo strategijo, ne opazujejo funkcije, ki preslikuje stanje okolja v akcije. Ne zapomnijo si, katera stanja so že videla in katere akcije so bile že izvedene. Sistemi s spodbujevanim učenjem, ki izkoriščajo funkcijo vrednosti, opazujejo in si zapomnijo tako stanja kot akcije.

Model okolja posnema okolje. Modeli so uporabljeni za načrtovanje, s katerim se določijo naslednje odločitve na podlagi tega, kakšne so pričakovane nagrade v naslednjih možnih stanjih. Tako se agent hkrati uči iz napak in iz modela okolja.

3.3 Markovska lastnost

V sistemu s spodbujevanim učenjem agentove odločitve temeljijo na signalu, ki prihaja iz okolja, ki ga imenujemo stanje okolja. Okolje je lahko opisano na

veliko načinov in lahko poda veliko podatkov. Ena lastnost močno definira, v kakšnem okolju smo. Ta lastnost se imenuje **markovska lastnost**.

Stanje je markovsko, če je naslednjo stanje odvisno le od trenutnega [9]. Na primer, pri igri šah markovska lastnost velja, saj trenutna pozicija figur vsebuje vso informacijo za odločitev. Pri problemu, kot je šah, ne potrebujemo zgodovine potez. V nobenem problemu pa ne potrebujemo več informacij kot jo vsebuje celotna zgodovina stanj. Markovska lastnost je koristna aproksimacija, ki poenostavi učne algoritme, saj nam ni potrebno hraniti zgodovine [9].

Markovski odločitveni problem (angl. Markov Decision Process) je problem, ki izpolnjuje pogoje markovske lastnosti. Če ima končno množico stanj in končno množico možnih akcij, potem se problem imenuje končni markovski odločitveni problem (angl. Finite Markov Decision Process).

3.4 Dinamično programiranje

Dinamično programiranje (angl. Dynamic programming, DP) vsebuje algoritme v spodbujevanem učenju, ki jih lahko uporabimo za izračun optimalne strategije, če je model okolja markovski odločitveni problem. Klasični algoritmi DP imajo omejeno uporabnost v spodbujevanem učenju, ker pričakujejo popoln model okolja in potrebujejo mnogo računske moči. Algoritmi DP so pomembni v teoriji spodbujevanega učenja. Ključna ideja DP je organizirati iskanje strategije s pomočjo funkcij vrednosti.

Optimalno strategijo lahko določimo, če imamo optimalno funkcijo vrednosti. Algoritmi DP so sestavljeni tako, da Bellmanove enačbe spremenimo v posodobitve za popravke napovedi vrednosti stanj. Funkcijo vrednosti lahko izračunamo, če imamo informacijo o ugodnih stanjih v okolju: tistih, ki vrnejo pozitivno nagrado, in tudi tistih, ki vrnejo negativno nagrado. Naslednji korak se imenuje popravljanje (angl. backing up), kjer si vsako stanje doda vrednosti svojih sosedov normalizirano s številom sosedov. Naloga se izvede n-krat, in sicer od 0 do razdalje n. Struktura enačbe je prikazana v

(3.1). Pomen simbolov v enačbi je: s - stanje, π - strategija, a - akcija, R - funkcija nagrade, A - množica možnih akcij glede na stanje, S - množica stanj in γ - faktor zmanjševanja. Takšni metodi rečemo tudi iterativna evalvacija strategije (angl. Iterative Policy Evaluation) [2].

$$V_{k+1}(s) \leftarrow \sum_{\alpha \epsilon A} \pi(a|s) \left[R(s,a) + \gamma \sum_{s' \epsilon S} P(s'|s,a) V_k(s') \right] \tag{3.1}$$

Da dobimo zaporedne približke vrednosti, metoda uporabi isto operacijo za vsa stanja na podlagi starih vrednosti stanj. V vsaki iteraciji iterativne evalvacije strategije »popravljamo« vrednosti vseh stanj, da dobimo približno funkcijo vrednosti V_{k+1}. Obstaja veliko različnih »popravkov«, odvisno od tega, ali popravljamo vrednost stanja ali kombinacijo stanja in akcije, in od tega, na kakšen način so kombinirane vrednosti zaporednih stanj. »Popravki« v DP so polni (angl. full back up) zato, ker delajo na osnovi vseh sosednjih stanj in ne le za vzorec naslednjih možnih stanj.

Ko izračunamo funkcijo vrednosti, lahko določimo optimalno strategijo. Trenutna strategija π agentu narekuje, katero akcijo naj izbere. Ko v nekem stanju izberemo akcijo, ki ni v strategiji, in potem primerjamo rezultat glede na strategijo, lahko izberemo učinkovitejšo strategijo. Učinkovitost izračunamo s pomočjo izraza (3.2). Pomen simbolov v enačbi so Q - vrednost akcije in r - nagrada.

$$Q^{\pi}(s,a) = E_{\pi}\{r_{t+1} + \gamma V^{\pi}(s_{t+1}) | s_t = s, a_t = a\} \tag{3.2}$$

Ključni kriterij je, da če neka sprememba v strategiji prinese boljši rezultat, potem je boljše izbirati spremembo. Opisan postopek je del teorema o izboljšavi strategije (angl. Policy improvement theorem) [24].

Evalvacija strategije (angl. Policy Evaluation) opisuje iterativno računanje funkcij vrednosti za dano strategijo. Izboljšava strategije (angl. Policy improvement) opisuje računanje dane funkcije vrednosti za izboljšano strategijo. Ko združimo metodi, dobimo iteracijo strategije (angl. Policy Iteration) in iteracijo vrednosti (angl. Value Iteration), zelo priljubljeni metodi v DP.

Eno ali drugo lahko uporabimo kot zanesljiv način, kako izračunati optimalno strategijo in funkcije vrednosti za končne markovske odločitvene probleme.

Metode DP opravijo »polne popravke« v vseh stanjih. Vsak popravek posodobi vrednost enega stanja glede na možna naslednja stanja. Ko »popravki« DP ne povzročajo več sprememb, je postopek končan.

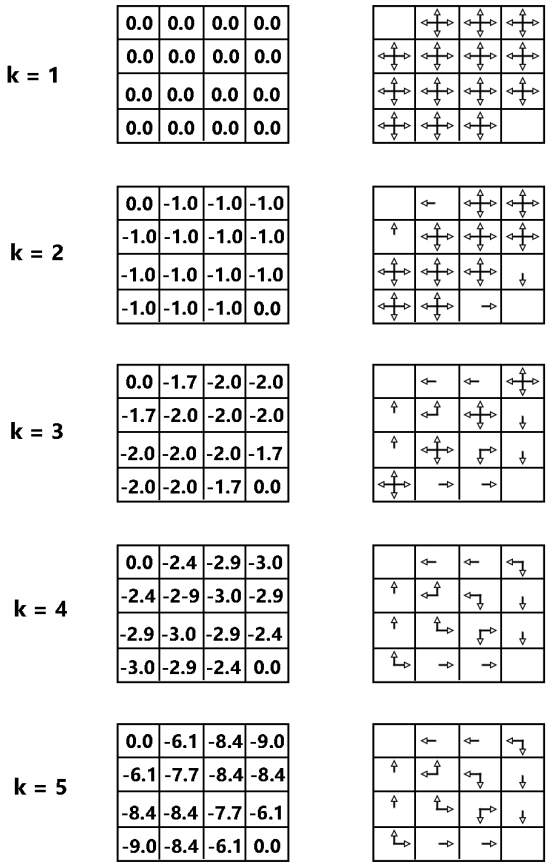
Ena pomembnih lastnosti metod DP je, da je ažuriranje napovedi za vrednosti stanj osnovano na napovedi vrednosti pomožnih naslednjih stanj. Takšen način pridobivanja napovedi imenujemo »zankanje«. Ključno pri DP je, da imamo natančen opis okolja in da se uporablja »zankanje«. Primer »zankanja« je razviden na sliki 3.2.

3.5 Metode Monte Carlo

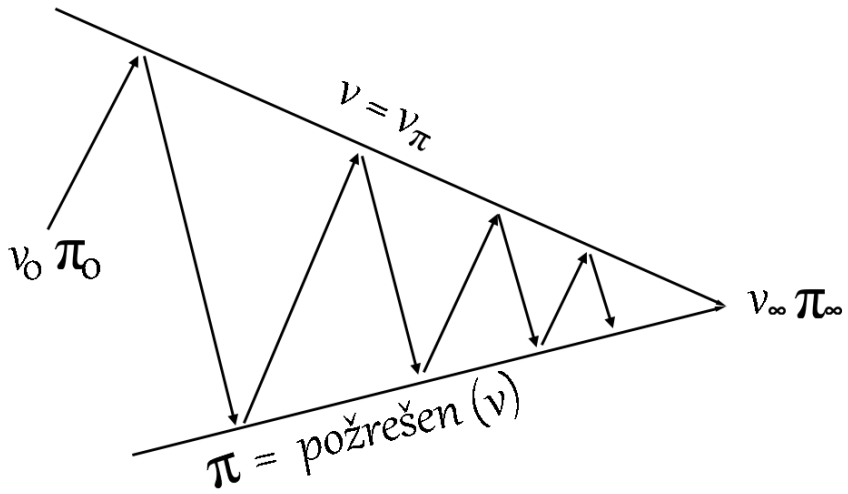
Z razliko od DP, Monte Carlo metode (MCM) potrebujejo le izkušnje, vzorce zaporedji stanj, akcij in nagrad v živo ali iz simulirane interakcije z okoljem. Učenje v živo ne potrebuje predhodnega znanja o okolju, a vseeno lahko doseže optimalno obnašanje. Čeprav MCM potrebujejo model okolja, mora model generirati samo vzorce prehodov in ne celotne verjetnostne distribucije vseh možnih prehodov, kot je potrebno pri metodah DP.

Pri MCM pričakujemo, da je problem epizoden (angl. Episodic). Pričakujemo, da je interakcija z okoljem razdeljena na epizode in da se vse epizode enkrat končajo ne glede na to, katere akcije so bile izbrane. Samo po koncu vsake epizode se spreminjajo napovedane vrednosti stanj in strategija. Izraz »Monte Carlo« je večkrat uporabljen za opis operacije, ki vključuje neko stohastično komponento. Tu jo uporabljamo izključno za metode, ki temeljijo na vzorčenju celotne pričakovane nagrade [12].

MCM razvijejo svoje funkcije vrednosti in optimalne strategije na podlagi izkušenj v obliki vzorčnih epizod. To pomeni, da imajo MCM tri prednosti pred metodami DP. Prva je, da se lahko naučijo optimalnega obnašanja direktno s stikom z okoljem brez modela okolja. Druga je, da jih lahko uporabljamo pri simulaciji ali vzorčenje modelov. Tretja je, da je MCM možno



Slika 3.2: Izgradnja funkcijo vrednosti v primeru dvodimenzionalnega sveta, ki vrača nagrado z vrednostjo -1 razen v levem zgornjem in desno spodnjem kotu, kjer vrača 0. Vrednost k predstavlja število iteracij. Na levi so vrednosti stanj, na desni pa strategija.



Slika 3.3: Procesa evalvacije strategije in izboljšave strategije medsebojno sodelujeta.

učinkovito osredotočiti na manjši podseznam stanj. Eno področje, ki nas bolj zanima, lahko natančno evalviramo, ne da bi natančno preverili ostala stanja. Obstaja še četrta prednost in to je, da so MCM manj občutljive na kršitve markovske lastnosti. Razlog je, da se tukaj ne ažurira vrednosti na osnovi vrednosti naslednjih možnih stanj. Tukaj se »zankanje« ne uporablja [1].

Posplošena iteracija strategije (angl. Generalized policy iteration, GPI) opisuje medsebojno delovanje procesov evalvacije strategije in izboljšave strategij. MCM so zgrajene tako, da sledijo shemi posplošene iteracije strategij. Ideja GPI je predstavljena na sliki 3.3.

MCM namesto uporabe modela okolja za izračun vrednosti vsakega stanja uporabijo povprečno pričakovano nagrado iz vsakega stanja posebej. Ker je vrednost stanja dolgoročno pričakovana nagrada, je lahko povprečje dober približek pravi vrednosti. Še posebej je uporabno približevanje funkcijam akcija-vrednost, saj te lahko izboljšajo strategijo brez modela okolja.

Problem pri MCM je vzdrževanje dovoljšnje stopnje raziskovanja. Ni dovolj, da izberemo akcije, ki so trenutno ocenjene kot najboljše, saj ostale akcije tako ne dobijo pričakovane nagrade. V tem primeru ne bi nikoli odkrili boljše izbire. Za rešitev tega problema obstajata dve metodi: vključena strategija (angl. On policy) in izključena strategija (angl. Off policy). V vključeni strategiji agent zmeraj raziskuje in izbere strategijo, ki zmeraj raziskuje. Tudi v izključeni strategiji agent raziskuje, nauči pa se deterministične optimalne strategije, ki mogoče ni povezana s strategijo, ki ji je sledil med učenjem [21].

Čeprav je med MCM in DP nekaj ključnih razlik, se najpomembnejše ideje ujemajo. Funkcija vrednosti se izračuna na enak način in tudi način iskanja optimalne vrednosti je enak.

3.6 Učenje s časovnimi razlikami

Ena centralnih idej spodbujevanega učenja je učenje s časovnimi razlikami (angl. Temporal Difference Learning, TDL). TDL gre za kombinacijo idej iz MCM in DP. Tako kot pri MCM se tudi pri TDL metode lahko učijo le iz izkušenj in brez modela okolja. Tako kot metode DP tudi metode TDL ažurirajo vrednosti, ki delno bazirajo na ostalih naučenih napovedih, brez čakanja na končni rezultat, torej se uporablja »zankanje« [8].

Metode TD (angl. Temporal difference) so alternativni način za reševanje problema vrednosti glede na MCM. Nadzorni problem pri obeh načinih izhaja iz posplošene iteracije strategije. Ideja je, da bi morali biti funkciji za približek strategiji in izračun vrednosti stanj medsebojno usklajeni tako, da se skupaj premikata proti optimalni vrednosti, kot je razvidno na sliki 3.3.

Eden izmed dveh procesov, ki skupaj sestavljata GPI, vodi funkcijo vrednosti, da čim bolj točno napove pričakovano nagrado za trenutno strategijo (problem napovedovanja). Drugi proces temelji na izkušnji, potrebno je vzdrževati dovoljšnjo količino raziskovanja [23].

Metode TDL niso vezane samo na spodbujevano učenje, lahko jih bolj

posplošimo. Primerne so za učenje dolgoročnih napovedi dinamičnih sistemov. TDL lahko uporabimo za napoved finančnih podatkov, življenjske dobe, volilnih rezultatov, vremenskih vzorcev, obnašanja živali ipd.

V naslednjem poglavju predstavimo, na kakšen način smo implementirali spodbujevano učenje z našim agentom v igri Knoxball.

Poglavje 4

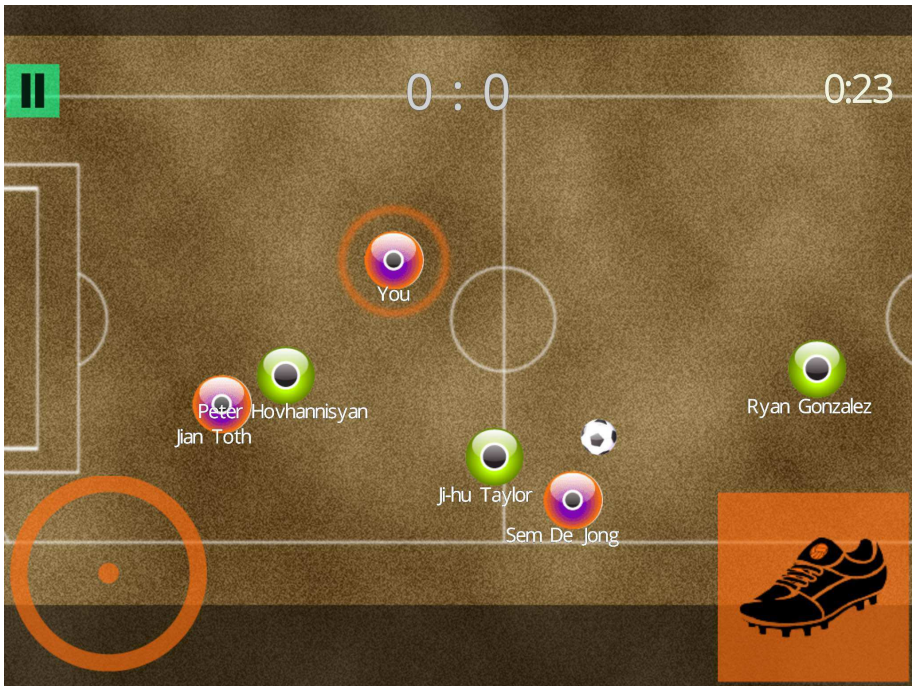
Agent s spodbujevanim učenjem v dinamični igri

Za primer uporabe spodbujevanega učenja na napravi z omejenimi viri smo izbrali mobilno igro. Izbrali smo mobilno igro Knoxball, ki smo jo razvili sami. Igro smo izbrali zato, da bi imeli proste roke pri implementaciji, saj imamo njeno izvorno kodo. Spodbujevano učenje smo implementirali tako, da pravilno deluje tudi na napravi z omejenimi viri.

V nadaljevanju opišemo koncept igre Knoxball, potem pa podrobneje strukturo in delovanje agenta s spodbujevanim učenjem, ki smo ga implementirali.

4.1 Opis igre

Knoxball je igra, narejena za mobilne naprave z operacijskim sistemom iOS. V celoti je napisana v programskem jeziku Objective-C. Igra poleg drugih standardnih knjižnic iOS uporablja knjižnico XNI, ki je podprta v iOS od inačice 6.1 naprej. Deluje tudi na tabličnih računalnikih z operacijskim sistemom iOS. Za platformo iOS (namesto za platformo Android ali Windows) smo se odločili zaradi programske opreme, ki je bila na razpolago, in trenutne razširjenosti platforme iOS. Potrebovali smo računalnik z operacijskim siste-



Slika 4.1: Primer igre Knoxball, ko imata obe ekipi po tri igralce.

mom OS X in z Xcode ter mobilni telefon z operacijskim sistemom iOS verzije vsaj 6.1. Za testiranje aplikacije na mobilnem telefonu smo potrebovali tudi univerzitetno licenco [11].

Knoxball je športna igra, mešanica med nogometom in zračnim hokejem. Izvaja se v realnem času. Bistveni del igre je tekma med dvema ekipama (vsaka ima svoje igralce – enega, dva ali tri), ki branita svoj gol. Obe ekipi imata skupno igrišče, na katerem se izvaja tekma, in žogo, ki jo morajo spraviti v gol nasprotnika. Ekipa, ki zabije več golov nasprotniku, zmaga. Igralci so predstavljeni s premikajočimi entitetami v obliki kroga, ki lahko poveča gibalno količino žoge, ko je ta dovolj blizu (strel). Imajo tudi možnosti premikanja žoge ob dotiku in premika naprej. Žoga se odbija od stranic. Izgled igre je viden na sliki 4.1.

V osnovi je igra enostavna. Kot pri pravem nogometu so za zmago proti dobri nasprotni ekipi potrebni razumevanja igre, dobre reakcije in razmišljanje vnaprej. Kakovostna igra je polna hitrih manevrov, podaj, strelov,

odbitih strelov, obramb, »dummy« potez, »preigravanja« in prerivanja. Za zmago obstaja veliko različnih strategij. To pomeni, da proti dobrim igralcem je težko zmagati.

Dober igralec se mora pravilno postaviti. Vedeti mora, kdaj je primeren čas za strel na gol, kdaj odbiti strel na gol ter kdaj preigrati nasprotnika. Vedeti mora tudi, kdaj se je bolje obnašati obrambno in kdaj je čas za napadalno igro.

Agent s spodbujevanim učenjem je za takšno igro primeren in koristen, saj bo agent končni avtomat vedno ponavljal iste napake, medtem ko se agent s spodbujevanim učenjem skozi čas izboljšuje. Knoxball predstavlja dober primer, kako je lahko spodbujevano učenje učinkovito pri ustvarjanju občutka igre proti pravemu človeku, sploh v primerjavi s šibkejšimi učnimi modeli.

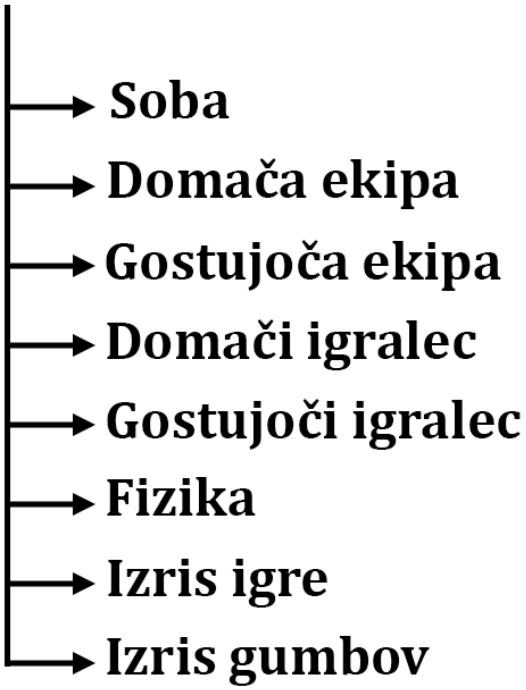
V naslednjem podpoglavju predstavimo sestavo igre, glavno zanko igre ter kako je v igro vključen agent s spodbujevanim agentom.

4.2 Arhitektura igre in agenta

Igro sestavlja več komponent, ki med seboj sodelujejo. Komponente se izvajajo zaporedno, ko so na vrsti v glavni zanki igre (angl. Game Thread). Glavne komponente igre in vrstni red izvajanja, je razviden iz slike 4.2. Zanka se izvede tridesetkrat na sekundo. Algoritmi agenta s spodbujevanim učenjem se izvajajo v vlogi domačega ali gostujočega igralca, odvisno od tega, v kateri ekipi je.

Če agent nima smeri, kamor bi se premaknil, se odloči za naslednji prostor, ki se po njegovi oceni najbolje splača. Če se agent premika in se njegova lokacija glede na diskretizacijo igrišča ne spremeni, ne dela nobenih dodatnih odločitev. Ko se agentu spremeni lokacija glede na diskretizacijo igrišča, se na novo odloči, kam se najbolj splača premakniti.

Ves čas si agent shranjuje seznam vseh stanj okolja, v katerih je bil. V trenutku zadetka agent dodeli stanjem, skozi katera je šel, vrednost glede



Slika 4.2: Seznam komponent igre in vrstni red, po katerem se izvajajo. Komponenta soba se ukvarja z dogodki, ki se dogajajo v igri: zgodil se je zadetek, konec igre, itd. Ekipi dodeljujeta vloge igralcem, če vsebujejo agente s končnim avtomatom. Komponenta fizika nadzoruje fiziko znotraj igre, da se žoga odbija iz stranic, da se vsi igralci pomikajo naprej glede na gibalno količino, itd. Komponenti za izrisovanje izrisujeta elemente na pravilno mesto na ekranu naprave.

na uspešnost. Agent te vrednosti vpiše v svojo funkcijo vrednosti. S temi podatki in ažurirano funkcijo vrednosti lahko agent izbira svoje odločitve.

Agent ima po zadetku in po vpisu vrednosti v funkcijo vrednosti možnost izvesti lokalno »zankanje«. Tukaj je veliko računanja, zato agent izvaja lokalno »zankanje« v ozadju. Zaradi ogromnega števila možnih stanj okolja ima agent po daljšem času učenja veliko podatkov. Izmerili smo, da je agent imel v nekaterih primerih več kot 100 MB podatkov in s tem podaljšal nalaganje podatkov na mobilno napravo. V času igranja pa nismo opazili upočasnitve igre zaradi agentov s spodbujevanim učenjem.

V naslednjem podpoglavju pregledamo, kako smo implementirali agenta s spodbujevanim učenjem, glavne komponente tega agenta in njegovo sestavo.

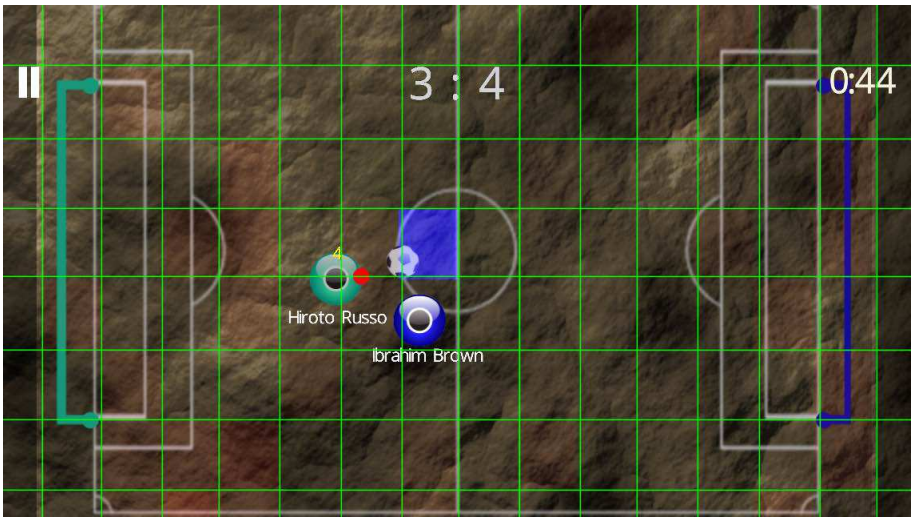
4.3 Uporaba metod spodbujevanega učenja v našem sistemu

V nadaljevanju opišemo glavne komponente našega agenta s spodbujevanim učenjem in sodelovanje med komponentami.

Ker smo uporabljali spodbujevano učenje na napravi z omejenimi viri, smo morali biti pozorni na porabo virov. Pri večini implementacij našega agenta s spodbujevanim učenjem smo se zgledovali po MCM, prisotnih pa je tudi nekaj idej iz DP in TDL. Naš problem ima približno markovsko lastnost, kar natančneje pokažemo kasneje.

4.3.1 Diskretizacija igre

Eden najbolj zahtevnih izzivov pri implementaciji agenta s spodbujevanim učenjem je diskretizacija igre. Osredotočili smo se na igro »ena na ena« (igralec proti igralcu). Poleg obeh igralcev je na igrišču še žoga. Za stanje okolja bi lahko upoštevali pozicijo, hitrost in smer igralca ter žoge in to, ali igralca streljata. Pozicija golov določa vrednost nagrade, ko je žoga v enem izmed golov.



Slika 4.3: Primer diskretizacije lokacij vseh igralcev in žoge.

Z uporabo samo pozicije obeh igralcev in žoge lahko skoraj zagotovimo markovsko lastnost za stanje okolja, vendar ne popolnoma, saj ne moremo razbrati smer gibanja in hitrost obeh igralcev in žoge. Končna optimalna strategija in funkcija vrednosti sta slabši, kot če bi upoštevali tudi hitrost in smer vseh elementov. Uporabljena diskretizacija prostora je vidna na sliki 4.3. Če vključimo tudi smer in hitrost premikov, dodamo toliko dodatnih stanj okolja, da je hitrost učenja zelo zmanjšana, čeprav bi bila to bolj optimalna rešitev. Toda že diskretizacija pozicij igralcev in žoge na sprejemljivo število stanj je, v primeru, ko si agent shrani vse podatke o teh stanjih, na meji zmogljivosti naprave z omejenimi viri.

Za diskretizacijo okolja smo tako določili lokacijo vseh treh elementov in stanje agenta, ali strelja ali ne. Vzdolžno koordinato (x) vseh lokacij smo razdelili na n-delov, navpično koordinato (y) vseh lokacij pa na m-delov. To, ali agent strelja ali ne, podvoji število stanj. Tako imamo na koncu možnih stanj $N_s = n^3 * m^3 * 2$. Večja kot sta n in m, boljša sta lahko najbolj optimalna strategija in funkcija vrednosti, s tem pa je tudi obnašanje agenta bolj optimalno. Toda z večanjem vrednosti n in m tudi upočasnimo hitrost učenja agenta, saj v istem času več stanj ostane neznanih. Uporabili smo vrednosti $n = 13$, $m = 8$ in tako dobili število možnih stanj $N_s = 2,249,728$.

4.3.2 Strategija

Strategijo uporabimo za določanje naslednje akcije, ki jo izvedemo glede na stanje okolja, v katerem je agent. Maksimalno število možnih akcij, ki jih ima na razpolago agent, je v našem primeru 16: 8 za smer, kar se podvoji z upoštevanjem, če agent strelja ali ne. Možne smeri so: gor, dol, levo, desno ter diagonalni premiki. Čas smo razdelili tako, da agent pregleda, ali je čas za novo odločitev ali ne šestkrat na sekundo.

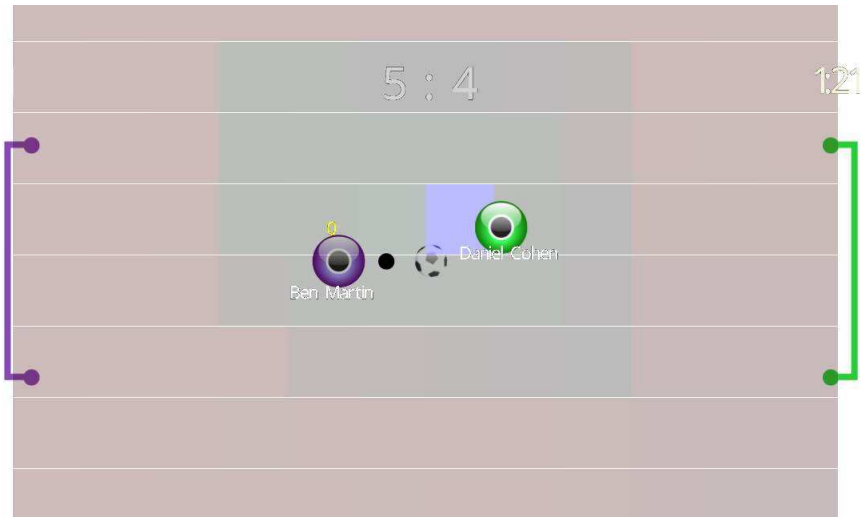
Vse možne akcije, ki jih lahko agent izvede, so dodane v seznam. Prihodno stanje izračunamo tako, da napovemo, kje bosta nasprotnik in žoga na igrišču glede na njuni gibalni količini v času, ko bo agent prispel do ciljne lokacije glede na izbiro stanja ter izbrano akcijo.

Seznam vseh akcij sortiramo glede na povprečno pričakovano nagrado za stanje, v katerega bi akcija igralca privedla. Vrednost q določa, s kakšno verjetnostjo bomo izbrali stanje z najvišjo napovedano pričakovano nagrado. Verjetnost, da se ne izbere najbolje ocenjena akcija, je $V_{a>1} = 1 - q$. Verjetnost, da ne izberemo niti prve niti druge akcije, je $V_{a>2} = (1 - q) * (1 - q)$ itd. Takšen način izbiranja naslednje akcije omogoča večjo fleksibilnost in možnost tako izkoriščanja kot raziskovanja, glede na trenutne potrebe in že dobljene izkušnje. V našem sistemu se lahko vrednost q veča glede na izkušnje.

4.3.3 Zapisovanje izkušenj

Zaradi uporabe MCM je naš problem razdeljen na epizode. Meje epizod so doseženi goli, tako agentovi kot nasprotnikovi. Med vsako epizodo agent shranjuje seznam stanj, v katerih se je nahajal do trenutka gola. Zadnja stanja so pomembnejša, medtem ko začetna stanja nimajo tako velikega vpliva.

Ko pripisujemo pričakovano nagrado stanjem v seznamu, z vsako iteracijo pričakovano nagrado pomnožimo s faktorjem zmanjševanja (γ). Dlje časa traja epizoda, manj pomembna so začetna stanja. Faktor zmanjševanja nagrade pomaga dodeliti čim bolj pravično pričakovano nagrado vsem sta-



Slika 4.4: Prikaz povprečne pričakovane nagrade za vsako stanje okolja z že določenimi lokacijami nasprotnika in žoge. Če je pričakovana nagrada negativna, bo polje obarvano rdeče, če pa je pričakovana nagrada pozitivna, bo polje zeleno. Večja kot je pričakovana nagrada, močnejša je barva.

njem tako, da dodeli manjšo nagrado tistim stanjem, ki jih je agent obiskal bližje začetku epizode.

Vrednosti stanj zapišemo v agentovo bazo stanj. Za vsako stanje zapišemo skupno vsoto pričakovane nagrade in kolikokrat je bil agent prisoten v tem stanju. Na podlagi tega vračamo povprečno pričakovano nagrado. Na tej podlagi se agent odloča, kako obetavno je neko stanje, kar je razvidno na slikah 4.4 in 4.5.

Opisana oblika zapisa stanja zagotavlja, da je okolje zelo blizu markovski lastnosti, saj ne potrebujemo zgodovino stanj okolja, da dovolj natančno določimo, kako ugodno je dano stanje okolja. Hkrati so jasna stanja, v katerih je pričakovana nagrada pozitivna ali negativna in hkrati zaključi epizodo: če je žoga na poziciji 0 ali $n-1$ je torej v golu. Zaradi te enostavne funkcije nagrade smo zapis združili s funkcijo vrednosti, ki vrača pričakovano nagrado glede na povprečje preteklih izkušenj.

Za pospešitev učenja agenta smo implementirali tudi lokalno »zankanje« tako, da na koncu vsake epizode za vsako stanje, ki je bilo prisotno v epizodi,

Slika 4.5: V tej situaciji vidimo veliko ugodnih stanj, saj je žoga blizu nasprotnikovega gola in bližje, kot je nasprotnik.

naredimo lokalno »zankanje« do stanj, ki so oddaljena največ b od omenjenega stanja. Tako pogladimo vrednosti stanj, ki so v okolju danega stanja in se izognemo temu, da bi se agent ujel v lokalne ekstreme.

V naslednjem poglavju bomo pogledali rezultate agentov in metodologijo njihovega ovrednotenja.

Poglavje 5

Evalvacija

S pomočjo preštetih golov in uporabo agentov brez učenja smo določili učinkovitost agentov v našem problemu.

Najprej opišemo agenta brez učenja, ki smo ga implementirali ter uporabili kot osnovo za ocenjevanje izboljšanih agentov. Predstavimo rezultate evalvacije in jih analiziramo.

5.1 Končni avtomati za evalvacijo učljivega agenta

Ekipe, ki so v celoti sestavljene iz agentov, ki so končni avtomati, so tudi same končni avtomati [3] in so lahko v stanju napada ali obrambe. Med igralci ni komunikacije, zato je možnost komunikacije z igralci dodeljena koordinatorju ekipe, ki med igro dodeljuje vloge igralcem. Glede na število igralcev so mogoče različne vloge. V izjemnih primerih lahko koordinator igralce tudi resetira [11].

Ko je v ekipi samo en igralec, mu je lahko dodeljena samo vloga vratarja. Ko sta v ekipi dva igralca, sta možni vlogi vratar in napadalec. V primeru treh igralcev je možna še vloga asistenta. Glede na vlogo ima igralec različen nabor stanj. Na nabor stanj vpliva tudi stanje ekipe. Ko je vratar v napadu, so njegova stanja sledeča: drži žogo, nima žoge, preigravanje, podaja, strel, zataknjen. Ko je vratar v obrambi, ima stanja: nima žoge, pokrij napadalca,

prestrezanje, rešuje gol, zataknjen. Napadalec ima podoben nabor, toda ne rešuje gola. Ko je ekipa v napadu, ima asistent samo stanje »nima žoge«. V trenutku, ko asistent dobi žogo, postane napadalec, prejšnji napadalec pa postane asistent. Ko je ekipa v stanju obrambe, ima asistent enak nabor stanj kot drugi igralci, razen reševanja gola.

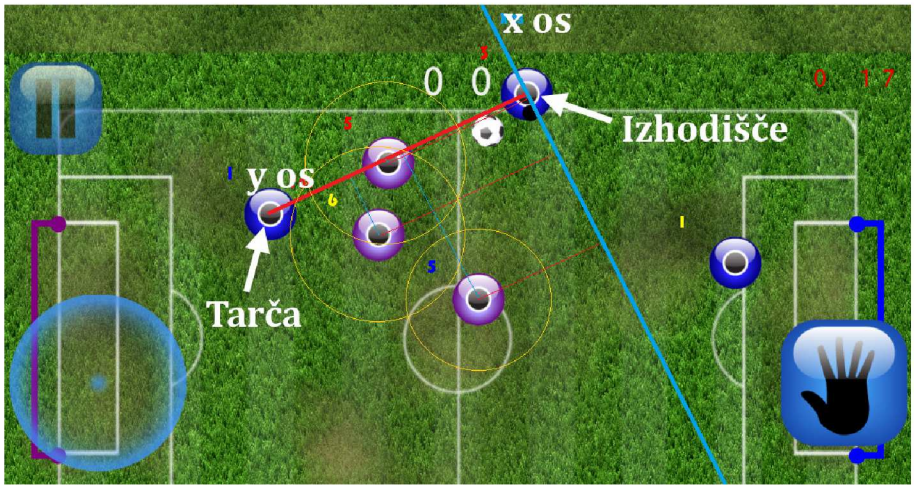
Igralce vodijo končni avtomati glede na cilje, ki so določeni za vsako stanje [3]. Vsako stanje sestoji iz enega ali več ciljev. Večina ciljev je preprostih, npr. premik proti žogi, prestrezanje, vrtenje okoli žoge, itd.

Nekatera stanja postanejo bolj kompleksna s kombinacijo več ciljev, ocenjevanj in izračunov. Pri podaji proti drugemu igralcu najprej izračunamo oceno, kako smiselna je taka podaja. Ocena sestoji iz treh delov: čas vrtenja okoli žoge, nevarnost prestrezanja igralcev nasprotne ekipe in lega soigralcev. Vsak del ocene ima utež, ki določa, kako pomemben je. Igralec oceni soigralce in ali jim je vredno podati. Če oceni, da ne, se odloči za preigravanje.

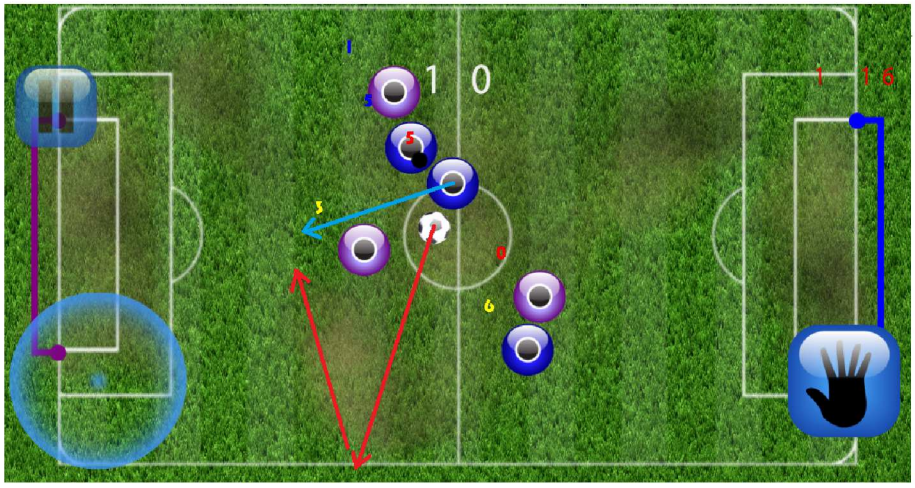
Zanimiv je algoritem za izračun nevarnosti prestrezanja žoge s strani igralcev nasprotne ekipe. Igralca, ki ocenjuje soigralce za podajo, preslikamo v lokalni koordinatni sistem glede na premico med podajalca in tarčo. Ostale pomembne dejavnike preslikamo v svoj lokalni koordinatni sistem, to so tarča (ki ima x-koordinato 0) ter vsi igralci iz nasprotne ekipe. Večja kot je y-koordinata pri nasprotnih igralcih, večji obseg kroga nevarnosti ima. Obseg kroga predstavlja čas, ki ga ima na razpolago igralec nasprotne ekipe za prestrezanje žoge. Igralci nasprotne ekipe z negativno vrednostjo y-koordinate niso pomembni, saj so v trenutku podaje postavljeni za igralcem z žogo in je ne morejo prestreči. Po oceni za nevarnost vsakega igralca nasprotne ekipe se posebej izračuna skupna nevarnost podaje. Grafična ilustracija izračuna je vidna na sliki 5.1.

Na podoben način se igralec odloči za strel. Določi si seznam točk na črti gola in oceni vsako točko posebej, če predstavlja ugodno tarčo. Na oceno vplivata samo čas vrtenja okoli žoge in nevarnost strela. Nevarnost strela računamo na enak način kot pri podaji.

Načinov preigravanja nasprotnika je več vrst. V trenutku, ko hoče igralec



Slika 5.1: Prikaz algoritma za računanje nevarnosti. V prikazenem primeru sta dva igralca nasprotne ekipe, ki bi lahko prestregla žogo. Modra črta prikazuje x os lokalnega koordinatnega sistema igralca, rdeča pa y os. Velikost rumenih krogov okoli nasprotnikov prikazuje območje, kamor bi se lahko premaknili glede na čas, ki ga imajo, preden gre žoga mimo njih.



Slika 5.2: Primer preigravanja. Igralec skuša pretentati nasprotnika tako, da odbije žogo od stene po poti rdeči puščici, sam pa potuje v smeri modre puščice, kjer kasneje sprejme žogo.

preigrati nasprotnika, si naključno izbere nek način preigravanja glede na to, katere pozna. Preigravanje največkrat sestoji iz več ciljev, ki se uresničujejo po vrsti. V primeru odboja od zida in nazaj je igralcu prvi cilj pripeljati žogo v bližino zida. Naslednji cilj je brcniti žogo tako, da se odbije od zida in nazaj k igralcu. Zadnji cilj je ponovno sprejeti žogo. Primer enega izmed načinov preigravanja je prikazan na sliki 5.2.

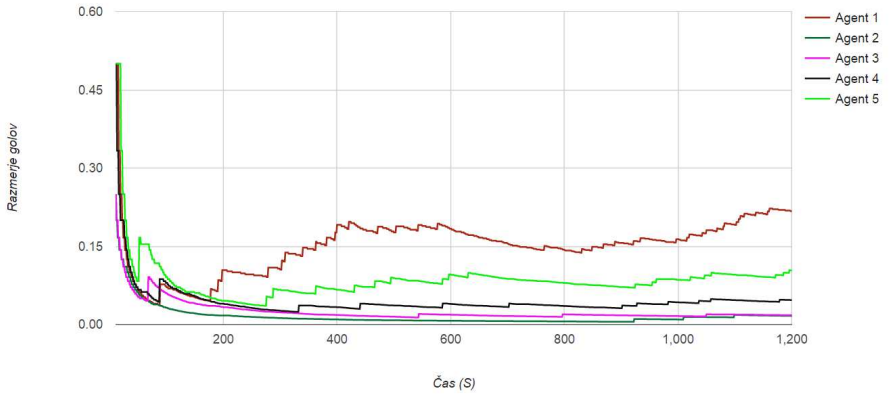
Čeprav je pri premikanju takšnega agenta dodano nekaj naključnosti, se agent ne uči, čeprav se ne bo vsakič enako obnašal. To daje agentom s spodbujevanim učenjem enake možnosti za uspeh glede na okolico.

5.2 Rezultati

Vsi uporabljeni agenti s spodbujevanim učenjem v začetku niso imeli nobenega predznanja. Vsi so imeli enake zmožnosti gibanja in učenja, spreminjali smo jih glede na vpliv »zankanja« iz sosednjih stanj, razmerje med raziskovanjem in izkoriščanjem ter faktor zmanjševanje nagrade γ.

Nasprotnik, s katerim smo merili uspešnost agentov, je bil za vse agente enak. Konfiguracije preizkušenih agentov so predstavljene v tabeli 5.1. Končni avtomat, ki se ne uči, bo zmeraj z neko verjetnostjo delal enake napake. Na sliki 5.3 vidimo razmerje golov z nasprotnikom do 1200. sekunde. Razmerje golov predstavlja količnik med agentovimi in nasprotnikovimi goli. V tem času nobenemu agentu ni uspelo zadeti več golov kot nasprotnik, kar je bilo pričakovano, saj končnega avtomata težko premaga tudi srednje dober človeški igralec.

Najboljše rezultate je dosegel Agent 1, ki je uporabil nekaj vpliva »zankanja«, imel dovoljšnje razmerje med raziskovanjem in izkoriščanjem ter zmeren faktor zmanjševanja nagrade. Tudi iz grafa je razvidno, da so vsi agenti dokaj hitro zadeli prvi gol in tudi v nadaljevanju redno zabijali gole nasprotniku. Rast razmerja golov pri vseh agentih kaže tudi na večjo učinkovitost glede na količino izkušenj.



Slika 5.3: Razmerje golov med agenti s spodbujevanim učenjem in končnim avtomatom. Agent 1 se je najbolje izkazal v tem časovnem okviru, lastnosti agentov so opisane v tabeli 5.1.

Tabela 5.1: Vrednosti parametrov agentov s spodbujevanim učenjem

Ime	Vpliv »zankanja«	Požrešnost strategije	Faktor zmanjševanja
Agent 1	30 %	80 %	5 %
Agent 2	30 %	100 %	5 %
Agent 3	0 %	80 %	5 %
Agent 4	30 %	80 %	0 %
Agent 5	90 %	80 %	5 %

Po več kot 100,000 obiskanih stanjih v bazi izkušenj (nekatera stanja so bila obiskana tudi do 1,000-krat) dveh agentov RL1 in RL2 smo ponovno preverili uspešnost agentov s spodbujevanim učenjem v primerjavi s končnim avtomatom in z agentom s spodbujevanim učenjem brez izkušenj. Rezultati so vidni na grafu 5.4, parametri agentov so opisani v tabeli 5.2. Graf, tako kot prejšnji, prikazuje razmerje med goli obeh ekip.

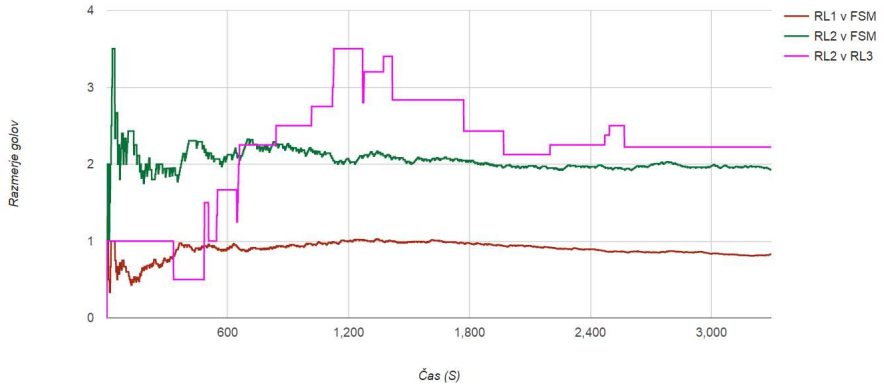
Pri rezultatih agenta RL1 proti končnemu avtomatu je razvidno, da je agent RL1 postal enakovreden nasprotnik agentu končni avtomat. Dolgo časa sta imela skoraj izenačen rezultat. Razlog za to, da agent RL1 v tem času ni prevladal, je, da v 20 odstotkih časa ne izbere najbolj ugodne akcije glede na svoje ovrednotenje. Na začetku nabiranja izkušenj je to sprejemljivo, saj vrednosti pričakovane nagrade še niso konvergirale. Po dovolj obiskih stanje konvergira in razvidno je, ali je ugodno ali ne. Torej ima agent v 20-odstotkov stanj možnost, da bo izbral slabšo akcijo. Tu je razvidno, kako velik vpliv ima izbira dobre strategije.

Agent RL2 je povsem prevladal nad agentom končnim avtomatom. Agenta RL1 in RL2 se razlikujeta samo v strategiji. Agent RL2 je dinamično spreminjal razmerje izkoriščanja in raziskovanja in pokazal, kako ključna je čim bolj optimalna implementacija tega razmerja v strategiji. Agenta RL1 in RL2 sta se sicer učila enako količino časa.

Rezultati igre agenta RL2 proti agentu RL3 so dokazali, da ima agent z več izkušnjami večje možnosti za zmago, vendar je bilo na tekmi manj golov, če sta oba agenta s spodbujevanim učenjem, kar smo pričakovali. Zaradi raziskovalne narave obeh agentov s spodbujevanim učenjem sta agenta tudi zaradi drug drugega veliko časa prebila v stanjih, ki jih nista poznala. To privede do naključnega obnašanja in več pretečenega časa med goli.

Pričakovana nagrada za stanja po nekem času konvergira v pravo vrednost stanja. Opazili smo, da po okoli 1100 obiskov stanja vrednost skonvergira do odstotka razdalje do končne vrednosti.

Opisane agente RL1, RL2, RL3 in FSM smo preizkusili še proti petim človeškim igralcem. Ocenili so kvaliteto obnašanja agenta - nasprotnika: ali



Slika 5.4: Razmerje golov med agenti. RL1 je agent s spodbujevanim učenjem z nespremenljivim razmerjem izkoriščanja in raziskovanja z vrednostjo 80 odstotkov. RL2 predstavlja agenta s spodbujevanim učenjem z dinamičnim razmerjem izkoriščanja, ki povečuje vpliv izkušenj s časom, glede na število obiskanih stanj. FSM je agent končni avtomat. RL3 predstavlja agenta s spodbujevanim učenjem, ki nima izkušenj.

Tabela 5.2: Parametri agentov iz slike 5.4

Ime	Zapisanih stanj	Požrešnost strat.	»zankanje«	Faktor zmanjš.
Agent RL1	>100,000	statična	30 %	5 %
Agent RL2	>100,000	dinamična	30 %	5 %
Agent RL3	0	statična	30 %	5 %

Tabela 5.3: Ocene in povprečni rezultati proti človeškim igralcem

Proti	Kvaliteta (/10)	Igralnost (/10)	Čas (min)	Zadetkov
Agent RL1	5,0	5,4	5,34	4,0
Agent RL2	7,4	7,8	6,58	5,8
Agent RL3	2,0	2,4	2,39	0,8
Agent FSM	6,8	6,8	2,45	7,6

je bil odziven in ali je reagiral, kot so pričakovali. Ocenili so tudi zabavnost ali igralnost igre z agentom. Zabeležili smo tudi rezultat in čas trajanja igre. Igro smo zaključili, ko je ena izmed ekip prva dosegla 10 zadetkov. Rezultati so prikazani v tabeli 5.3. Prikazane vrednosti so povprečne ocene in rezultati vseh petih človeških igralcev. Razvidno je, da je bil agent RL2 najbolj uspešen pri zabijanju golov človeškim nasprotnikom med vsemi agenti s spodbujevanim učenjem. Tekme so bile najdaljše ravno tako pri agentu RL2, ocenjen je bil kot najbolj zabaven in kakovosten. Agent RL3 je bil, pričakovano, najmanj zanimiv in učinkovit.Tekme so bile kratke in dobil je najslabše ocene.

Izmed vseh agentov je bil glede na rezultat najbolj učinkovit agent predstavljen s končnim avtomatom. To je bilo pričakovano, saj samo agent s končnim avtomatom pozna pravila igre. Problem je v tem, da so tako agenti s spodbujevanim učenjem kot tudi človeški igralci nepredvidljivi, kar velikokrat povzroči, da je agent s spodbujevanim učenjem v stanju okolja, ki ga ne pozna. To prinaša naključno obnašanje, čeprav imajo agenti s spodbujevanim učenjem, ki imajo več izkušenj bolj optimalno strategijo tudi boljši rezultat. Agenta s spodbujevanim učenjem bi lahko še izboljšali, če bi veliko izkušenj nabral tudi proti človeškim igralcem in ne samo proti agentu s končnim avtomatom.

Preizkuševalci so povedali, da je agent s končnim avtomatom velikokrat ponavljal svoje obnašanje in s tem postal predvidljiv.

V celoti gledano je razvidno, da več časa, kot se agent s spodbujevanim

učenjem uči, bolj bo uspešen. Pomembno vlogo pri večji uspešnosti ima tudi strategija.

Poglavje 6

Sklepne ugotovitve

Implementirali smo inteligentnega agenta z uporabo algoritmov spodbujevanega učenja v mobilni igri z omejenimi viri. Igra se je izvajala na mobilnem telefonu. Pokazali smo, da agent s časom postane bolj vešč in bolj nevaren nasprotnik. Po dovolj dolgem času učenja se je agent lahko enačil s končnim avtomatom ali ga celo prekašal.

Agent, implementiran z algoritmi spodbujevanega učenja, je zmožen v kratkem času najti načine za doseganje golov in s tem doseči svoje cilje. Celoten sistem se brez težav izvaja na napravah z omejenimi viri. Sistem ni zahteven in veliko računanja se izvaja v ozadju, kar pripomore k hitremu in gladkemu delovanju igre.

Ostaja več možnih izboljšav sistema. Funkcija vrednosti bi lahko izboljšali tako, da bi hitreje konvergirala ali s pomočjo polnega »zankanja« v ozadju ali s pomočjo nadzorovanega učenja. Trenutno do konvergence preteče precej časa, saj je možnih stanj ogromno. Izboljšava bi terjala več računskih virov, česar smo se v tej nalogi izognili.

Učenje bi lahko združili z nadzorovanim učenjem in tako zmanjšali število stanj s pomočjo učenja podobnosti stanj. V tem primeru bi bilo veliko število stanj manj problematično in bi lahko dodali še nove informacije o okolju in s tem dosegli skoraj popolno markovsko lastnost.

Celotnega agenta s fleksibilnim načinom opisovanja signalov okolja bi

lahko preizkusili tudi v primeru igre z več igralci: »dva na dva« ali »tri na tri«. Takšno fleksibilnost bi lahko preizkusili tudi v drugih igrah.

Kot zaključek povemo, da vseh težav ni mogoče rešiti zgolj z vnaprej danimi informacijami in z nadzorovanim učenjem. Treba je pridobiti ključne izkušnje. Agenti, ki so se zmožni izboljševati v realnem času z nabiranjem izkušenj, so bolj robustni kot tisti, ki tega ne zmorejo. Spodbujevano učenje je zato obetavno orodje za vrsto aktualnih in zanimivih problemov, kombinacija spodbujevanega in nadzorovanega učenja bi lahko rešila tudi nekatere izmed najtežjih problemov v robotiki in umetni inteligenci.

Literatura

[1] A. Barto and M. Duff. Monte carlo matrix inversion and reinforcement learning. In *In Advances in Neural Information Processing Systems*, number 6, pages 687–694. Morgan Kaufman, 1994.

[2] D. Bertsekas. *Constrained Opimization and Lagrange Multiplier Methods*. Academic Press, Inc., 1982.

[3] M. Buckland. *Programming Game AI by Example*. Wordware publishing inc., 2005.

[4] J. Bura. *AI in Games*. Apress Media L.L.C., 2012.

[5] L. Floridi and J. Sanders. On the morality of artificial inteligence. *Mind and Machines*, 14(3):349 – 379, August 2004.

[6] H. Gelernter, J. Hansen, and D. Loveland. Empirical explorations of the geometry theorem machine. *American Federation of Information Processing Societies*, pages 143–149, May 1960.

[7] D. Johnson and J. Wiles. Computer games with intelligence. In *IEEE International Conference on Fuzzy Systems*, pages 1355 – 1358. Citeseer, 2001.

[8] A. Klopf. Brain function and adaptive systems: A heterostatic theory. Technical Report 79, Air Force Cambridge Research Laboratories, March 1972.

[9] I. Kononenko and M. Robnik Šikonja. *Inteligentni sistemi.* Založba FE in FRI, 2010.

[10] J. McCarthy. *Programs with common sense.* Defense Technical Information Center, 1963.

[11] D. McPartlin. *Asinhron večigralski način igranja mobilne igre preko medmrežja: Diplomsko delo.* McPartlin, D., 2013.

[12] D. Michie and R. Chambers. Boxes: An experiment in adaptive control. *Machine Intelligence*, 2:137 – 152, 1968.

[13] I. Millington and J. Funge. *Artificial Inteligence for Games (Second Edition).* Morgan Kaufman, 2009.

[14] M. Minsky. Steps toward artificial intelligence. *Proceedings of the IRE*, 49(1):8 – 30, January 1961.

[15] M. Minsky and S. Papert. Progress report on artificial intelligence. Technical report, University of Oregon, 1974.

[16] S. Russel and P. Norvig. *Artificial Inteligence - A modern Approach (Third edition).* Pearson Education, 2010.

[17] A. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210 – 229, July 1959.

[18] A. Saygin, I. Cicekli, and V. Akman. *Turing Test: 50 years later.* Springer Netherlands, 2003.

[19] W. Schultz, P. Dayan, and P. Montaque. A neural substrate of prediction and reward. *Science*, 275:1593 – 1599, March 1997.

[20] E. Shortliffe. *Computer-Based Medical Consultations: MYCIN.* American Elsevier Publishing Co., Inc., 1976.

[21] S. Singh and R. Sutton. Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22:123 – 158, 1996.

[22] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 1998.

[23] J. Van Der Wal. Discounted markov games: Generalized policy iteration method. *Journal of Optimization Theory and Applications*, 25(1):125 – 138, May 1978.

[24] C. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992.